We propose, through a study of the MACSYMA system, to determine appropriate strategies to attach data-type checking capabilities to LISP functions whose arguments are, for example, rational numbers, polynomials, well-formed ~~formulas~~ expressions ~~of one sort or another~~, etc. Separation of the data type from the implementation would also be pursued. Thus if the first element of a list L is the main variable of a polynomial, (CAR L) extracts that value, but means that no checking of appropriateness of the access can be done. However, if (MAINVAR L) = (CAR L) where L is of polynomial type, ~~the checking may be~~ type checking may be requested. Of course, if one wishes fast execution, no checking need be done, and (MAINVAR L) will be expanded at compile time to (CAR L), ~~and~~ and run at full speed. Procedures which construct data structures would have similar definitions.

Proofs of the correctness of algebraic programs would be of interest in that the algebraic programs might then be used, in a sense as "lemmas", to prove the correctness of other programs. The more substantial mathematical and ~~log~~ logical content of these programs, in spite of their size, suggest these as strong possibilities.

integrate

$$\frac{(x+1)\log(x+1) + x\log x}{x^2+x}.$$

Prove $\int \frac{\log(x+1)}{x}$ is not integrable.

June 7. L. Wigton

Conversion of given series to continued fractions

the idea is to expand the radius of convergence of a series, and accelerate convergence.

Consider $\arctan(x) = x - x^3/3 + x^5/5 - x^7/7 + - \ldots$

Since $\arctan(z) = \int_0^z \frac{dt}{1+t^2}$

there are singularities at $\pm i$, and the series is no good for $|x| \geq 1$.

We can try to accelerate series by various schemes, since, e.g.

$\frac{\pi}{4} = \arctan(1) = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + - \ldots$ converges very slowly

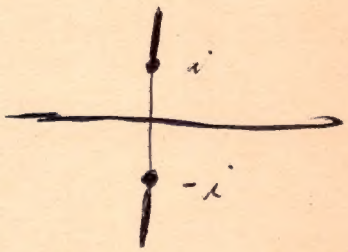
$$\arctan(x) = \cfrac{x}{1 + \cfrac{1x^2}{3 + \cfrac{(2x)^2}{5 + \cfrac{(3x)^2}{7 + \ldots}}}}$$

this can be unwound successively to get a rational expression

$$\frac{x}{1+\frac{x^2}{3}} = \frac{3x}{3+x^2} \Rightarrow$$

$$\arctan(x) \approx \frac{15x + 4x^3}{15 + 9x^2}.$$

This converges except along branch cuts of arctan.



The first few iterations of the continued fraction gives much higher accuracy e.g. for $\frac{\pi}{4} = \arctan(1)$, 10^{th} term $= .78539816$ > 8 significant figures.

Another example.

Recall $Ei(x) = \int_x^\infty \frac{e^{-t}}{t}\,dt = \frac{e^{-x}}{x}\left(1 - \frac{1!}{x} + \frac{2!}{x^2} - + \ldots\right)$

an asymptotic series.

~~To compute~~

To determine where to cut off the non-convergent series

$$F(x) = 1 + \frac{1!}{x} + \ldots$$

halt when the term is $\frac{n!}{n^n}$ for argument n.

If we use Stirling's formula, the error is $\sqrt{2\pi n}\,(n/e)^n/n^n$

$$= \sqrt{2\pi n}\, e^{-n}$$

$E_1(n)$

n	Abs. Error for summing n terms (best choice)
1	1
2	.5
5	.04
10	.00035
20	2.3×10^{-8}
30	1.3×10^{-12}

This is good for large n, but poor for small n, for which we could use Taylor series. Unfortunately there is a large gap (say between 3 and 20) where neither approximation is good. One way to bridge this gap economically is to use continued fractions.

Returning to the above $F(x)$, which is a power series in $\frac{1}{x}$,

$$F(x) = \cfrac{1}{1 + \cfrac{1/x}{1 + \cfrac{1/x}{1 + \cfrac{2/x}{1 + \cfrac{2/x}{1 + \cfrac{3/x}{1 + \ddots}}}}}}$$

Q. What if you approximate a terminating truncated power series (that is, a polynomial) by a continued fraction: the TPS has no error - but continued fraction does! Answer: It uses the idea of a Padé approximation which will find a rational approximation, which for many functions is better (but not all functions).

The continued fraction for 10 iterates is good to 8 figures of $E_1(10)$ or about twice the best truncated asymptotic series.

Computing $F(1)$ using continued fractions to successively higher iterates yields

$1.000,\ .500,\ .6666,\ .5714,\ \ldots\ .59738$ (10th term) $\ldots\ .596347363$

Computing a C.F. from power series is an $O(n^2)$ operation using standard methods, which makes it comparable to inversion (classical) of power series.

The Padé approximation idea is to find P, Q

by finding $\dfrac{P(x)}{Q(x)} = \sum_{i=0}^{n} a_i x^i = A$

but $P(x) = Q(x) * A$

gives a set of equations for coefficients, taking $O(n^3)$.

To find n successive approximations this way would take

$O(1^3 + 2^3 + \cdots + n^3) = O(n^4)$.

Fortunately there is a better way: "Quotient-Difference Algorithm"

$q_1^{(i)} = a_{i+1}/a_i \qquad i = 0, \ldots n$

Q. What if $a_i = 0$? There is no C.F. of standard form, but there is another. E.g. for $\sin$, we have a C.F. in x^2.

Compute a tabular array

$$\begin{array}{c|ccccc}
 & q_1^{(0)} & & e_1^{(0)} & & \\
0 & & q_1^{(1)} & & q_2^{(0)} & \\
 & q_1^{(1)} & & e_1^{(1)} & & e_2^{(0)} \\
0 & & q_1^{(2)} & & q_2^{(1)} & \\
 & q_1^{(2)} & & e_1^{(2)} & & e_2^{(1)} \\
0 & & q_1^{(3)} & & q_2^{(2)} & \\
 & q_1^{(3)} & & e_1^{(3)} & & \\
0 & & q_1^{(4)} & & &
\end{array}$$

where we use

$e_k^{(n)} + q_k^{(n)} = q_k^{(n+1)} + e_{k-1}^{(n+1)}$

to calculate e's

$q_{k+1}^{(n)} \cdot e_k^{(n)} = q_k^{(n+1)} \cdot e_k^{(n+1)}$

to calculate the q's

Bugs

1) ~~Bugs (??)~~ Locally declared arrays not permitted

2) weight x=1;
   ostlevel 5;

   {pow * pow now truncated ok}

   pt: = sub(x=t, pow) {makes series in t}

   sub(t=pow, pt } {messes up}

3) can't differentiate with respect to x (a weighted variable)

4) It appears, from the time taken, that the work is done, and then truncated afterward.

Oscar Lanford
- getting Math dept involved in MACSYMA
9th floor

The top line, $q_1^{(0)}$, $e_1^{(0)}$ ... represent continued fraction:

$$F(x) = \cfrac{q_0}{1 + \cfrac{-q_1^{(0)} \cdot x}{1 + \cfrac{-e_1^{(0)} x}{1 + \cfrac{-q_2^{(0)} x}{1 + \cfrac{\ddots}{}}}}}$$

The computational labor is $O(n^2)$

Example: (from $Ei(x)$ ~~example~~ again)

$\Sigma n!(-x)^n$. here $q_1^{(n)} = -(n+1)$

	-1			
0		-1		
	-2		-2	
0		-1		-2
	-3		-3	
0		-1		
	-4			

So C.F. = $\{1, 1, 2, 2, \ldots\} \equiv$

$$\cfrac{1}{1 + \cfrac{1 \cdot x}{1 + \cfrac{1 \cdot x}{1 + \cfrac{\ddots}{}}}}$$

Basic routines were programmed in Hearn's Reduce (version July 1976)

problems encountered:

```
For i:=1:10 do begin
    a:= i+1;
    if a>5 then goto out;
    a:= a+2;
    end;
    a:= 1+a
out:
```

(defect of some LISP: compared to algol - no jumps out of a loop)